НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“ ” 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки ***6.050103 «Програмна інженерія»***

на тему *Програмний інтерфейс системи оцінки ризиків*

Виконав: студент IV курсу, групи *ІП-з51в Козел Олена Олександрівна* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Баклан І.В.* _____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Рецензент:

доц. каф. ОТ, к.т.н., доц. Болдак А.О.
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

АНОТАЦІЯ

Пояснювальна записка до дипломного проекту містить 59 сторінок, графічних матеріалів, таблиць, рисунків.

Об’єкт розробки: програмний інтерфейс системи для оцінки ризиків.

Мета розробки: полегшення та систематизація процесу оцінки ризиків компаній аудиторами.

В наш час існує безліч підприємств. Невід’ємною частиною діяльності будь-якого підприємства є ризик. Технологічні, виробничі, комерційні, фінансові, природно-екологічні та інші – кожен із зазначених чинників тим чи іншим чином має вплив на будь-яке підприємство. На сьогодні, задля визначення, оцінки та контролю ризиків підприємства звертаються до аудиторських компаній. Саме вони допомагають підприємствам завчасно перебачити або ж підготуватися до проблем в разі їх виникнення і нашою метою є полегшення систематизації процесу оцінки та звітування компаній-аудиторів.

Розробка даної системи призначена для надання можливості аудиторським компаніям в зручному вигляді створювати універсальні шаблони для оцінки ризиків компаній-клієнтів. У дипломному проекті описані деталі проектування, розробки та тестування системи, наведені існуючі альтернативні варіанти таких систем, визначено необхідні інструменти та засоби, що потрібні для її створення та підтримки.

Пояснювальна записка складається зі вступу, 4 розділів, висновків, переліку посилань та додатків.

Ключові слова: ПРОГРАМНИЙ ІНТЕРФЕЙС, СИСТЕМА, ФРЕЙМВОРК, ВЕБ-ДОДАТОК, ЗАГРОЗА, ФАКТОР, КОНТРОЛЬ, МЕТОДОЛОГІЯ, ОЦІНКА, РИЗИК.

ABSTRACT

Explanatory note to the diploma project contains 59 pages, graphics, tables, figures.

Object: program interface for the risk assessment system.

The purpose of development: to ease and organize the process of risk assessment held by auditing companies.

Nowadays, there are many enterprises. An integral part of any enterprise is the risk. Technological, industrial, commercial, financial, natural-ecological and other - each of these factors in one way or another has an impact on any enterprise. Today, for the purpose of determining, assessing and controlling the risks of an enterprise, they are turning to audit firms. It is them, who help enterprises to redeem or prepare themselves for issues in case of their occurrence, and our goal is to facilitate the systematization of the process of evaluation and reporting to audit companies.

The development of this system is intended to enable auditing companies to create convenient and universal templates for assessing the risks of client companies. The diploma project describes the details of the design, development and testing of the system, presents the existing alternatives to such systems, identifies the necessary tools and required for its creation and support.

An explanatory note consists of an introduction, 4 sections, conclusions, a list of references and annexes.

Keywords: PROGRAM INTERFACE, SYSTEM, FRAMEWORK, WEB-APPLICATION, THREAT, FACTOR, CONTROL, METHODOLOGY, ASSESSMENT, RISK.

Пояснювальна записка
до дипломного проекту

на тему: Програмний інтерфейс системи для оцінки ризиків

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Загроза – обставини або події, що виникають у середовищі, які можуть бути причиною порушення політики безпеки інформації і (або) нанесення збитків автоматизованій системі.

Фактор – обставина, яка сприяє виникненню небезпечної ситуації (ризику).

Контроль – збірка факторів, яка має певне значення, що впливає на фінальну оцінку.

Методологія – список правил та формул, за якими проводиться оцінка ризиків.

Тип оцінки – шаблон, який включає в себе набір загроз, факторів , контролів та методологію, за якими буде проводитися оцінка певних ризиків.

Наслідуваний ризик – обставини або події, які відбуваються в середовищі та мають вплив на безпеку компанії-клієнта.

Протидія ризику – заходи та засоби для зменшення впливу певного ризику.

Залишковий ризик – результат протидії наслідуваному ризику.

Власник – користувач, який має можливість створювати сутності, створювати і конфігурувати шаблони.

Підписник – користувач, який може використовувати створені власником шаблони з метою оцінки ризиків клієнта.

Клієнт – компанія, яка замовляє послугу оцінки ризиків.

SaaS – модель поширення програм споживачам, при якій постачальник розробляє веб-програму, розміщує її й управляє нею (самостійно або через третіх осіб) з метою використання її замовниками через інтернет. Замовники платять не за володіння програмами як такими, а за їх використання (через API, що доступне через веб і яке часто використовують веб-служби). Близьким до терміну SaaS є термін «On-Demand» (за запитом).

Принциповою відмінністю моделі SaaS від раніших (Hosted Applications та Application Service Provider (ASP)) є те, що отримується саме послуга та інтерфейс (призначений для користувача або програми), тобто деяка функціональність без жорсткої прив'язки до способу її реалізації.

ВСТУП

Нестабільний стан економіки та несприятлива динаміка розвитку створює високий ризик для компаній. Організації стають уразливими і з’являється необхідність звертатися до нових підходів запобігання та керування ризиками. Таким чином, неймовірно важливим в життєвому циклі компаній стає комплексний аудит ризиків.

Варто зазначити, що рівень розвитку аудиторських компаній у всьому світі на сьогоднішній день, нажаль, не встигає за технологічним розвитком. Тобто, більшість процесів, конче необхідних для проведення якісного та детального аналізу та аудиту, досі використовують застарілі методи та обладнання.

Як ми знаємо, технології та автоматизація впроваджуються в кожній сфері людської діяльності, але, в деяких з них, нажаль, досить комфортного рішення ще не існує. Так, як аудиторська оцінка зазвичай п отребує врахування великої кількості чинників і для різних оцінок необхідні різні дані, дуже важко уявити та створити програмне забезпечення, яке враховувало б всі фактори та показники. Тому для нас стало метою створити рішення, яке допомогло б аудиторським компаніям проводити оцінку ризиків більш комфортно та з меншою ймовірністю пропустити важливі показники через людський фактор. За спроби знайти схожі рішення, позитивного результату ми не отримали.

В той же час, в мережі можна знайти багато компаній, які пропонують послуги аудиторської оцінки, що означає, що ринок для використання автоматизованої системи оцінки ризиків існує і є досить великим.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Веб-застосунок – клієнт – серверний застосунок, в якому клієнт взаємодіє із сервером за допомогою браузера, а за сервер відповідає веб-сервер. Логіка веб-застосунку розподілена між сервером та клієнтом, зберігання даних відбувається, зазвичай, на сервері, обмін інформацією відбувається по мережі. Однією з переваг є той факт, що клієнти не залежать від певної операційної системи користувача, тому веб-застосунки вважаються між платформними службами.

Оцінка ризику – встановлення (ідентифікація) відповідності загрози і можливих її джерел, дослідження механізмів їх появи, оцінка ймовірності появи небезпечних подій та їх наслідків. Оцінка ризику є компонентом аналізу ризику, відносних до якісного та попередньо якісному опису ризику і його складових елементів при зіставленні, розробці та реалізації нових об’єктів, а також на етапі визначення та встановлення строків безпечної експлуатації діючих об’єктів.

1.2 Аналіз успішних ІТ-проектів

На даний момент не існує універсального веб-застосунку, який міг бути використаний для різних типів компаній, різних типів оцінки ризиків за різними методологіями. Тому даний проект вважається унікальним.

1.2.1 Аналіз відомих технічних рішень

Перед створенням програмного інтерфейсу було проаналізовано декілька можливих варіантів. В результаті було обрано мову JavaScript, бібліотеки React, Redux та Thunk, а також утиліту Enzyme для написання тестів.

React — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту

веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників. [1]

React було обрано через його легкість у застосуванні, компонентну архітектуру та можливість створювати універсальні модулі. Його використовують для створення масштабних веб-застосунків, саме чим і має бути універсальна система для оцінки ризиків. Адже вона має обробляти велику кількість типів оцінки ризиків, які вираховуються за певними методологіями та включають в себе загрози, фактори, контролі та ризики. Також система буде використовувати дані, котрі оновлюються в реальному часі. Завдяки спеціальній архітектурі React, система буде оновлюватися без перезавантаження сторінки. Головною метою бібліотеки є швидкість, простота та масштабованість. React опрацьовує лише користувацький інтерфейс у застосунках. Він грає роль виду у паттерні (MVC), і може використовуватися в поєднанні з іншими JavaScript бібліотеками. Як бібліотеку користувацького інтерфейсу React зазвичай використовується разом з іншими бібліотеками, такими як Redux.

Redux - відкрита JavaScript бібліотека, інструмент для керування станом даних та станом інтерфейсу в JavaScript-застосунках. Як вже вказувалося, найчастіше використовується разом з React або Angular для побудови інтерфейсів користувача. [2] В React не рекомендовано реалізовувати пряму взаємодію між компонентами. Це вважається поганою практикою, приводить до помилок та спагетті-коду. Саме тут необхідний Redux. Він пропонує зберігати всі стани застосунку в одному сховищі. Компоненти відправляють свій стан в сховище, а не прямо іншим компонентам. Компоненти, які мають «знати» про ці зміни, моніторять сховище. Нижче зображено головний принцип роботи Redux:

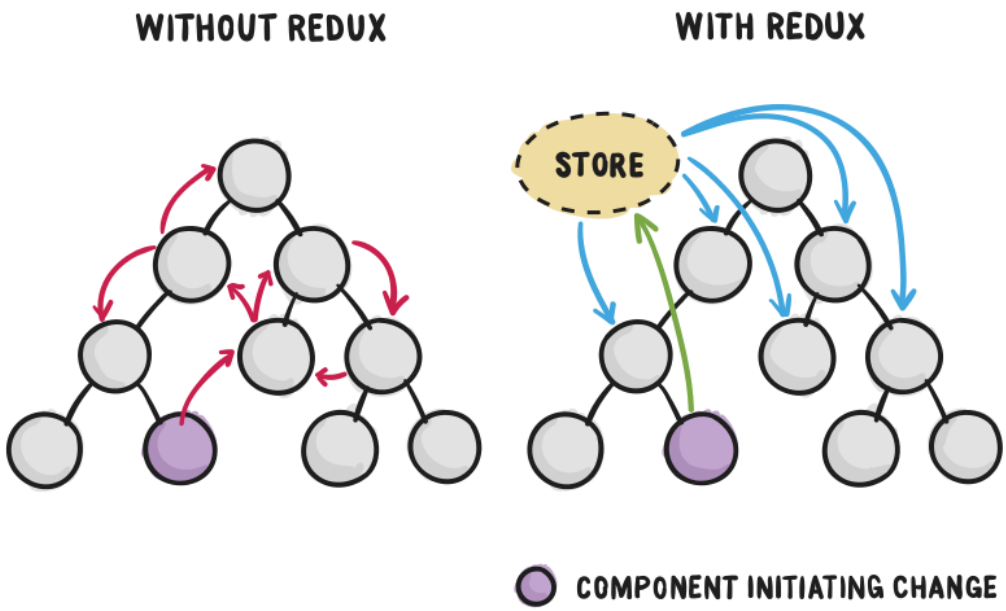

Рисунок 1.1 – Схема роботи Redux Thunk

Thunk – відкрита JavaScript бібліотека, яка використовується для налаштування асинхронних подій. За замовчуванням, творці подій в Redux не підтримують асинхронні події такі, як отримання даних, саме тому ми використовуємо Redux Thunk. [3]

Thunk дозволяє нам створювати події, які повертають функцію замість самої дії. Ця внутрішня функція може приймати методи сховища в якості параметрів.

Enzyme - це утиліта тестування JavaScript для React, яка полегшує тестування виводу компонентів React. Можна також керувати, проходити, і в деяких ситуаціях імітувати виконання коду з виведенням результату. [4]

1.2.2 Аналіз відомих програмних продуктів

На момент написання дипломної роботи, в мережі Інтернет можна знайти тільки обмежені пропозиції на дану тему. Наприклад, деякі компанії, які пропонують свої послуги, потенційно мають програмне рішення, але не видають його у відкритий доступ. Користувачі можуть звернутися до даних компаній, і вони будуть надавати власні послуги, користуючись своїм рішенням.

Інший варіант, який можна побачити в мережі – продукт, який доступний лише для покупки. Отже, по суті, користувачі не можуть спробувати застосувати продукт в своїй компанії, а тільки потім купити. Також, варто зазначити, що жодне із існуючих рішень не може бути використане в універсальному плані – для різних типів компаній та типів оцінки ризиків. До того ж, на ринку немає варіантів, які дозволяють купувати підписку, яка дає доступ до всього програмного забезпечення і може бути кастомізована під кожного користувача.

Пропоную розглянути вищеописані варіанти, оцінити їх характеристики, позитивні сторони та недоліки. Так, як більшість продуктів-конкурентів не мають вільного доступу, будемо їх аналізувати за описом.

a) Peregrine-International

https://www.peregrine-international.com/

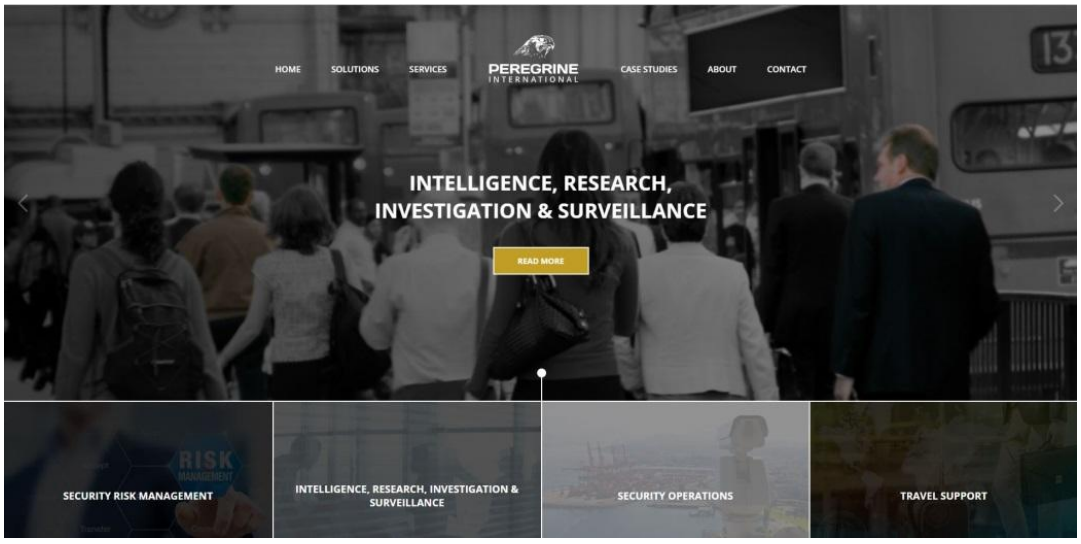

Рисунок 1.2 – Peregrine-International

Особливістю данного ресурсу є те, що користувач не має доступу до інформації про програмне забезпечення, яке використовується компанією. Натомість, власники сайту пропонують замовити послуги компанії, яка спеціалізується безпосередньо на роботі з ризиками безпеки, інтелектуальної власності та логістичними ризиками.

До позитивних сторін даного ресурсу можна віднести той факт, що компанія гарантує індивідуальний підхід до кожного замовника, що є дуже важливим зважаючи на велику кількість сфер напрямку підприємств.

Також, слід звернути увагу на те, що компанія не лише допомагає з визначенням та керуванням певних ризиків, але і пропонує окрему послугу – навчання працівників попереджати та правильно готуватися до впливу окремих ризиків. Але, нажаль, таке рішення підходить лише для невеликого прошарку організацій.

б) Alyne

https://www.alyne.com

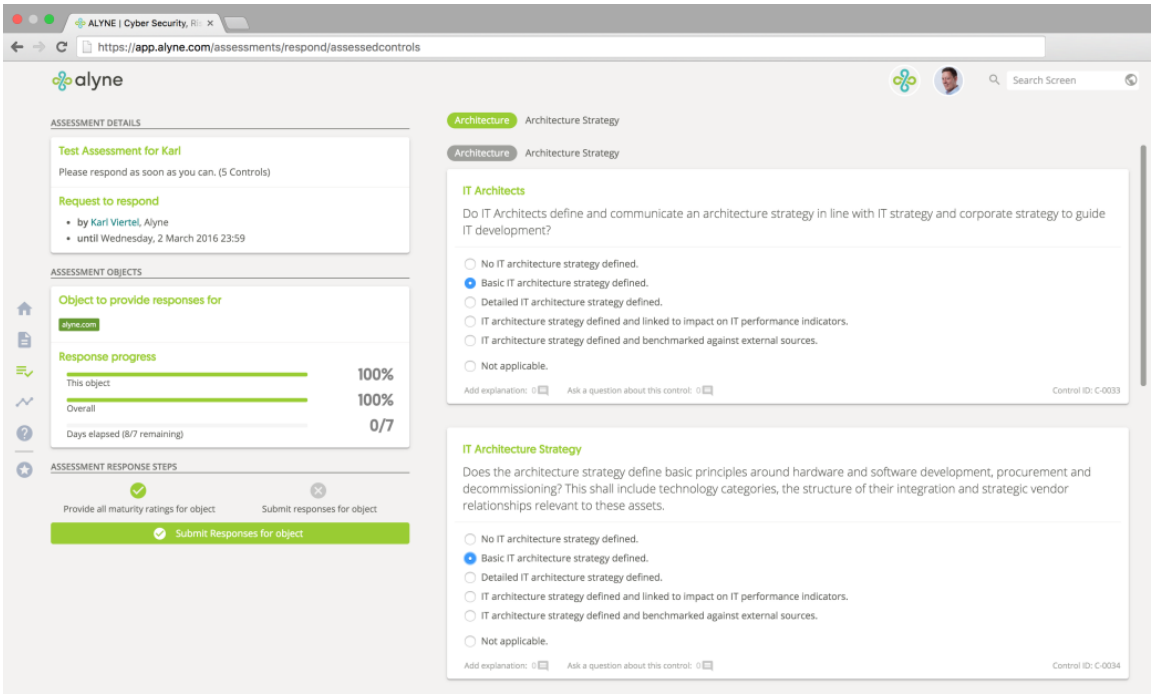

Рисунок 1.3 - Alyne

Даний ресурс, так же як і попередній, не надає можливості вільного доступу до самого програмного забезпечення, але все ж таки пропонує користувачеві більше інформації про себе. Продукт являється SaaS веб-застосунком, який дозволяє керувати своєю кібер-безпекою та ризиками.

Згідно з інформацією на офіційному сайти ресурсу, головними перевагами є наявність загального контенту, корисного всім сферам промисловості, який допомагає приймати зважені рішення.

Щодо негативних сторін даного застосунку, я б хотіла звернути увагу на те, що інтерфейс ресурсу хоч і є зручним, але дуже заплутаним. Адже судячи зі знімків екрану, наданих на сайті, ресурс має дуже багато опцій, які можуть бути не зрозумілими та можуть деізнформувати користувачів.

в) Plan Brothers – audits.io

https://www.planbrothers.io/products/audits

Згідно з інформацією на сайті, дана компанія надає послуги в різних напрямах, зокрема в аудиті ризиків. Головні переваги, виокремлені на самому сайті представляють собою:електронні чек-листи, прозоре звітування, зручне керування кореспонденцією, можливість прикріпляти картинки та файли, інтеграції, автоматична статистика та ін.

Щодо того, що можна віднести до негативних сторін ресурсу, я б хотіла безперечно зауважити, що дане програмне забезпечення допомагає лише з поверховим контролем ризиків, але не може забезпечити конкретність та точну оцінку всіх ризиків, можливих для всіх компаній-клієнтів.

г) Tracker Networks – ERM Essential

https://trackernetworks.com/

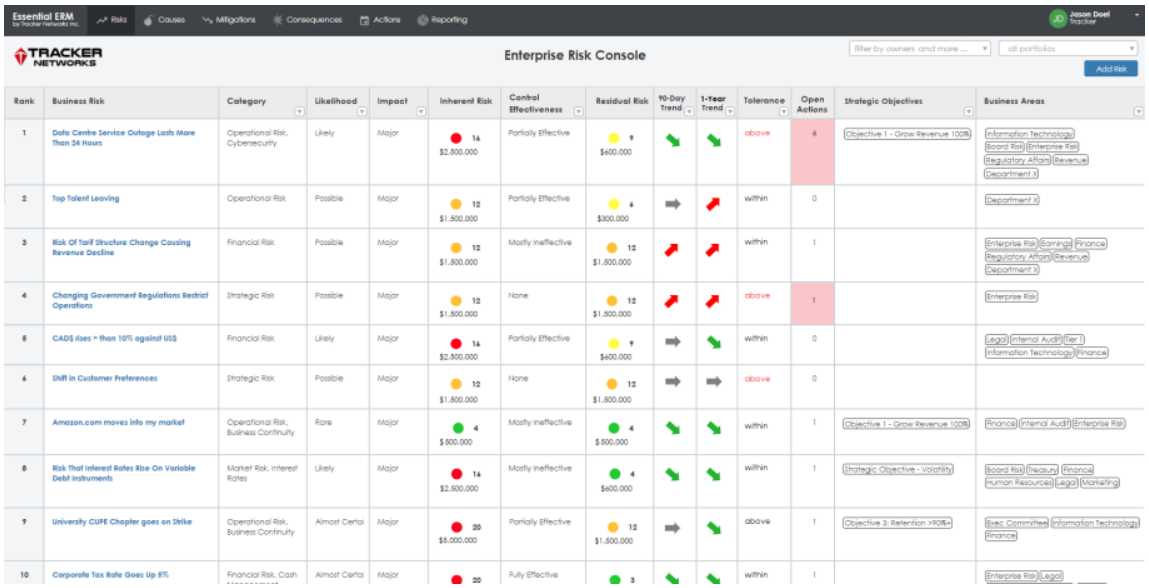

Рисунок 1.4 - ERM Essential

За інформацією, наданою сайтом головними перевагами є те, що даний програмний за стосунок простіше, ніж електронна таблиця з потужністю

автоматизованої системи. Він надає можливість класифікувати свої ризики в інтуїтивному середовищі перетягування та легко фільтрувати ризики за об'єктами, бізнес-областю, власником тощо.

Власне, до недоліків варто віднести обмеженість доступу до функціоналу сайту зважаючи на опис видів підписок. Тільки найдорожча підписка має всі необхідні для оцінки ризиків фактори, що змушує користувачів платити на всі опції, навіть якщо половина с них насправді не буде потрібна. Також, на сайті користувачі не мають змоги побачити реальну ціну продукту і мають додатково зв’язуватися з власниками продукту перед тим, як придбати ПЗ.

д) Orcanos

https://www.orcanos.com/compliance

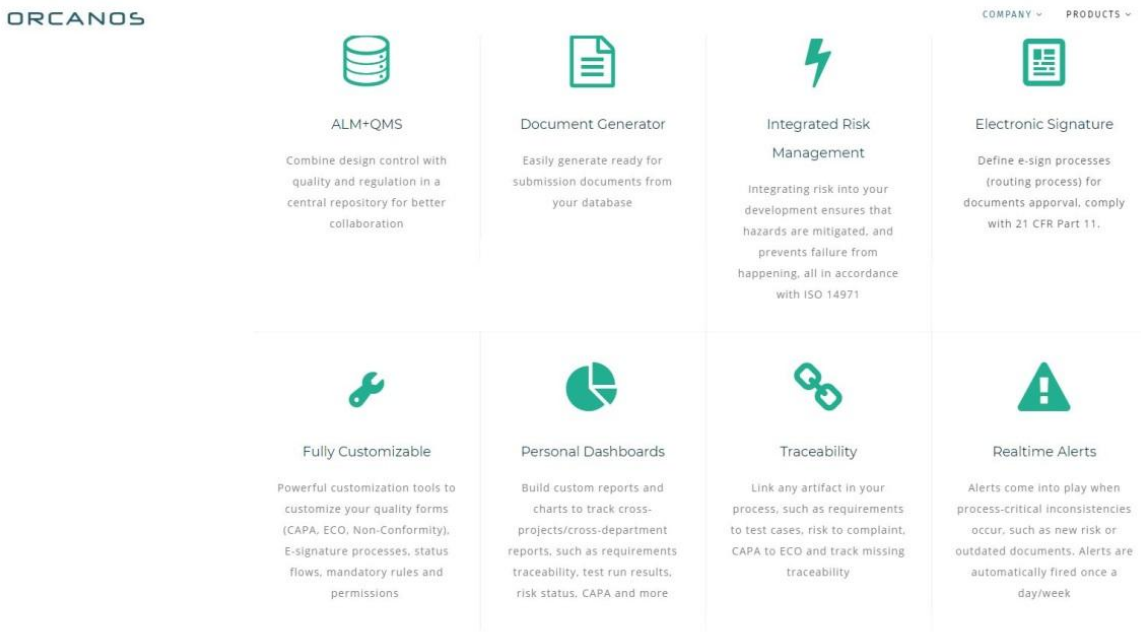

Рисунок 1.5 - Orcanos

Даний ресурс пропонує набір різних інструментів, які окремо можуть частково домогти з певними факторами, але не представляють собою цілісного рішення для оцінки та визначення методів запобігання різним ризикам.

В основному, даний веб-сайт надає можливість користуватися такими опціями як відслідковування дефектів, система керування якістю, контроль змін, інструменти для керування вимогами і т.д.

1.3 Аналіз вимог до програмного забезпечення

Акторами є: Гість, Власник, Системний Адміністратор, Підписник, Система.

Визначимо, які дії або варіанти використання вони виконують в системі, для цього наведемо таблицю 1.1, в якій описані актори, варіанти використання та їх описи дій.

Таблиця 1.1 - Типи залежностей між варіантами використання

Актор	Варіант використання	Опис дії варіанта використання
Гість	Авторизація	Після відкриття веб-застосунку система надає форму авторизації за допомогою якої користувач має змогу увійти до системи.
Системний адміністратор	Додати власника	Після натиснення кнопки «Додати» система надає форму для заповнення інформації про нового власника.
Системний адміністратор	Заповнити інформацію про власника	Система надає можливість додати ім’я, ел. адресу та рівень дозволу (редагування, права власника, права підписника)
Системний адміністратор	Переглянути інформацію про власника	Після натискання на поле власника, система надає

		інформацію про існуючого власника.
Системний адміністратор	Відправити запрошення	Після заповнення форми про нового власника, на його ел. пошту відправлено запрошення до системи.
Власник	Переглянути запрошення	Після того, як системний адміністратор відправив запрошення, користувач має відкрити та переглянути запрошення.
Власник	Прийняти запрошення	Після отримання та перегляду запрошення власник може прийняти запрошення.
Власник	Авторизація	За посиланням у запрошенні підписник може увійти у систему.
Власник	Додати підписника	Власник має змогу додати нового підписника, заповнивши необхідну інформацію про нього та надіславши запрошення.
Власник	Редагувати підписника	В додатку є кнопка, за допомогою якої власник може редагувати інформацію про підписника.

Власник	Створити тип оцінки ризику	Система надає змогу власнику створити тип оцінки ризику для кожного окремого підписника.
Власник	Редагувати тип оцінки	Система надає змогу редагувати створену раніше оцінку.
Власник	Архівувати тип оцінки	Система надає змогу пересунути обраний тип оцінки ризику до архіву.
Власник	Створити фактори	Система надає змогу власнику створити фактор для кожного окремого ризику.

Продовження таблиці 1.1

Власник	Редагувати фактори	Система надає змогу редагувати створену раніше оцінку.
Власник	Створити контроли	Система надає змогу власнику створити контрол для кожного окремого ризику.
Власник	Редагувати контроли	Система надає змогу редагувати створений раніше контроль.
Власник	Створити ризики	Система надає змогу власнику створити ризик для кожної окремої загрози.

Власник	Редагувати ризики	Система надає змогу редагувати створений раніше ризик.
Власник	Архівувати фактори	Система надає змогу пересунути обраний фактор ризику до архіву.
Власник	Архівувати контроли	Система надає змогу пересунути обраний контрол ризику до архіву.
Власник	Архівувати ризики	Система надає змогу пересунути обраний ризик до архіву.
Власник	Створити загрозу	Система надає змогу власнику створити загрозу для кожного окремого типу оцінки.
Власник	Редагувати загрозу	Система надає змогу редагувати створену раніше загрозу.
Власник	Архівувати загрозу	Система надає змогу пересунути обрану загрозу до архіву.
Власник	Сортувати елементи	Система надає можливість власнику сортувати елементи списку від А до Я або від Я до А.
Власник	Використовувати пошук	Система надає можливість

		використовувати пошук за назвою, акронімом або тегом.
Підписник	Обрати тип оцінки	Система надає змогу обирати тип оцінки серез визначених власником наперед типів
Підписник	Заповнити шляхи подолання впливу ризиків	Система надає змогу обрати необхідні для певної компанії-клієнта покрокові шляхи подолання ризиків
Підписник	Визначити рівень впливу наслідуваного ризику	Система надає змогу ознайомитися з потенційним рівнем впливу наслідуваного ризику
Підписник	Переглядати лист оцінок	Система надає змогу підписнику переглядати лист оцінок доступних для цієї конкретної компанії-клієнта
Підписник	Переглядати звіти	Система надає змогу підписнику переглядати фінальні звіти, отримані після заповнення всіх етапів оцінки ризиків
Підписник	Редагувати звіти	Система надає змогу підписнику редагувати

		значення фінальних звітів, отриманих після заповнення всіх етапів оцінки ризиків
Підписник	Обирати загрозу	Система надає змогу обирати та встановлювати значення заздалегідь визначених загроз
Підписник	Вказати значення контролу	Система надає змогу встановити певне значення обраного контролу ризику
Підписник	Вказати значення фактору	Система надає змогу встановити певне значення обраного фактору ризику

Відповідно визначених варіантів використання побудуємо загальну модель варіантів використання, яка наведена на КПІ.ІП-4410.045490–08–99.СС.

1.3.1 Розроблення функціональних вимог

а) Гість має змогу авторизуватися в системі.

1) Гостю надається форма для авторизації.

2) Система надає гостю можливість увійти в систему після вдалої перевірки введених даних.

б) Власник має змогу додавати підписника.

1) Система надає форму для заповнення інформації підписника.

2) Власник має змогу ввести інформацію підписника (електронну адресу, ПІБ, номер телефону, ліміт загроз, типи оцінки ризиків).

3) Власник має змогу ввести інформацію про компанію підписника (назва компанії, електронна адреса компанії, фізична адреса компанії, номер телефону компанії).

4) Система надає змогу відмінити або запросити підписника до користування системою.

в) Власник має змогу переглядати та редагувати лист підписників.

1) Власнику надається інтерфейс роботи з вкладкою «Лист підписників».

2) Власнику надається можливість пошуку та редагування підписників.

г) Власник має змогу створювати оцінки ризику.

1) Власнику надається інтерфейс для налаштування нової оцінки ризику – інтерфейс для вибору загроз, контролів та факторів.

2) Після вибору необхідних загроз, контролів та факторів та натискання кнопки «Зберегти» система зберігає налаштовану оцінку ризиків.

д) Власник має змогу редагувати оцінки ризику.

1) Власнику надається інтерфейс для налаштування нової оцінки ризику – інтерфейс для вибору загроз, контролів та факторів.

2) Після редагування необхідних загроз, контролів та факторів та натискання кнопки «Зберегти» система зберігає редаговану оцінку ризиків.

е) Власник має змогу архівувати оцінки ризику.

1) Власнику надається інтерфейс для налаштування нової оцінки ризику – інтерфейс для вибору загроз, контролів та факторів.

2) Після натискання кнопки «Архівувати» система запитує підтвердження і архівує оцінку ризиків.

ж) Підписник має змогу переглядати звіти.

1) Підписнику надається до перегляду сторінка зі звітом.

2) Підписник має змогу переглядати та працювати зі звітом.

з) Підписник має змогу працювати із листом оцінок.

1) Система надає до ознайомлення лист оцінок.

2) Підписник може обрати певний тип оцінки.

3) Після того, як підписник обирає тип оцінки, йому надається інтерфейс для заповнення інформації для формування звіту.

и) Користувач може використовувати пошук.

1) Система надає інтерфейс для пошуку показників (ризиків, факторів, методологій, контролів) за ім’ям, акронімом або тегом.

Таблиця 1.2

1.3.2 Розроблення нефункціональних вимог

До вихідного продукту пред’являються нефункціональні вимоги до:

– застосування (якість інтерфейсу, продукту й ін.);

– продуктивності (пропускна здатність, час реакції й ін.);

– зручність використання (інтуїтивна зрозумілість системи, наявність підказок та ін..);

– надійності виконання (без помилок і відмов) зовнішніх інтерфейсів, за якими виконується взаємодія з іншими компонентами або підсистемами.

1.3.3 Постановка комплексу завдань модулю

Даний модуль має виконувати наступні функції:

– надати інтерфейс для авторизації;

– надавати доступ до системи відповідно до наданих дозволів;

– додавати дані користувача до бази даних на етапі створення власника або підписника;

– створювати/редагувати/видаляти/архівувати типи оцінки до певних методологій;

– створювати/редагувати/видаляти/архівувати фактори до певних типів оцінки;

– створювати/редагувати/видаляти/архівувати контроли до певних типів оцінки;

– створювати/редагувати/видаляти/архівувати теги до певних типів оцінки;

– створювати/редагувати/видаляти/архівувати загрози до певних типів оцінки;

– сортувати елементи за алфавітом у прямому або зворотному порядку;

– відправляти запрошень новим користувачам;

– відновлювати елементи;

– заповнювати додаткові показники (вірогідність, вагу).

1.4 Висновки по розділу

У цьому розділі було проаналізовано відомі технічні рішення та програмні продукти, виявлено ознаки якісного продукту для даної предметної області. Головними недоліками існуючих рішень є відсутність повного доступного продукту, який не вимагає сторонніх компаній під час безпосереднього використання системи.

Щодо можливих технологій, було проведено аналіз і обрано мову JavaScript, призначену для створення інтерфейсів. Детальніше, було обрано фреймворки React, Redux та Redux Thunk, на основі якого і відбувається написання коду для реалізації інтерфейсу системи.

Головними вимогами щодо інтерфейсу продукту є зрозумілість, правильна організація та розміщення елементів для компактного розміщення складних елементів системи. Контент та опис розділів системи має бути простим та зрозумілим користувачам, важлива наявність візуальних вказівників та покрокових інструкцій. Система має бути адаптивною для можливості використання на більшості популярних пристроїв: мобільних девайсах, планшетах та браузерах (Google Chrome, Opera,Mozilla Firefox, Safari).

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Мобільний застосунок або мобільний додаток — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Більшість мобільних додатків встановлюються виробником на етапі виробництва але їх також можна завантажити та встановити самостійно з “магазинів”, таких як Google Play Store на пристроях Android, App Store на iOS, Windows Phone Store на мобільних пристроях з операційною системою Windows. Мобільні додатки можуть бути як безкоштовними, так і платними.

На сьогодні, мобільні застосунки мають широкий спектр використання - ігри, мобільні версії додатків соціальних мереж, GPS, браузери та багато іншого, хоча до цього вони використовувалися, в основному, для перевірки електронної пошти та повідомлень. Тому, можна сказати, що ринок мобільних застосунків сьогодні досяг високого рівня і невпинно зростає.

На рисунку 2.1 можна побачити діаграму бізнес-процесів розроблюваного застосунку.

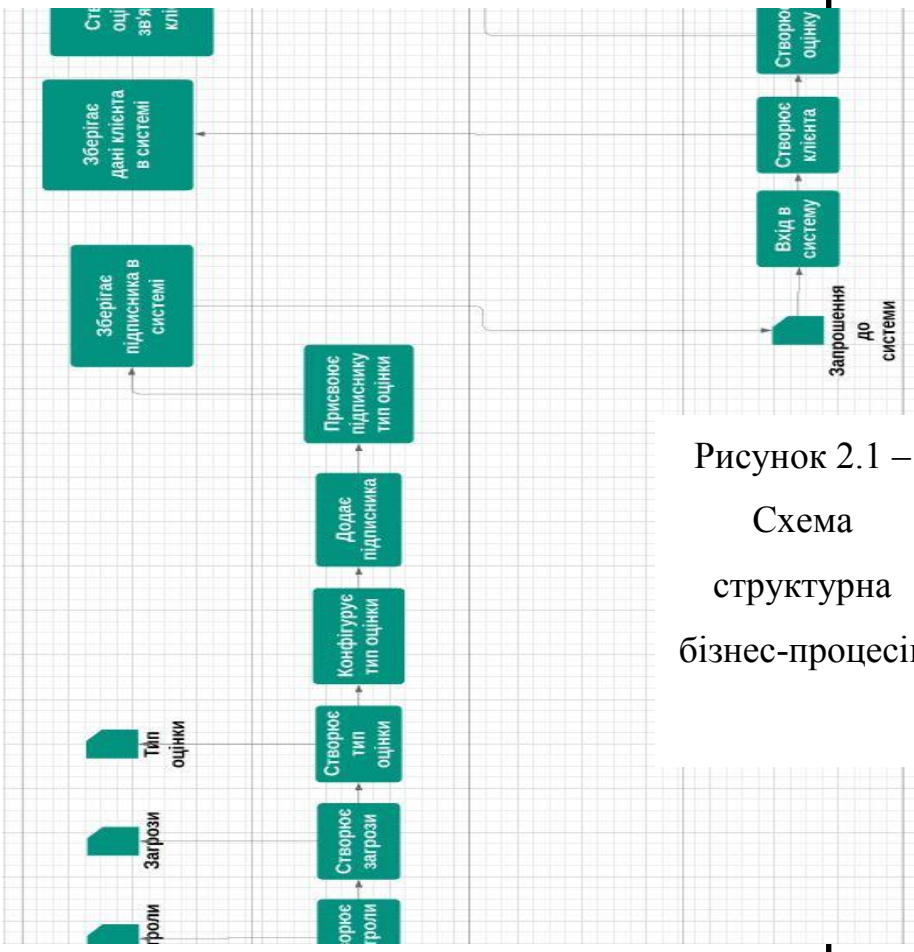

Рисунок 2.1 – Схема структурна бізнес-процесі

Розробка програмного забезпечення для мобільних пристроїв потребує врахування їх обмежень та можливостей. Мобільні пристрої працюють на акумуляторі та мають менш потужні процесори, ніж персональні комп'ютери, а також мають більше функцій, таких як визначення місцезнаходження та камери. Розробникам також доводиться враховувати широкий спектр розмірів екрана, різні технічні характеристики та конфігурації обладнання через сильну конкуренцію мобільного програмного забезпечення та зміни в кожній платформі.

2.2 Архітектура програмного забезпечення

Інтерфейс системи було створено за допомогою архітектури Redux.

Redux – це бібліотека для організації архітектури застосунку. Redux позиціонує себе як передбачуваний контейнер стану для JavaScript застосунків. Redux був натхненний ідеями Flux[5] та Elm[6] та зібрав найкраще з них.

Flux-архітектура — архітектурний підхід або набір шаблонів програмування для побудови користувацького інтерфейсу веб-застосунків, які поєднуються з реактивним програмуванням і побудований на однонаправлених потоках даних.

Elm (з англ. B'яз) — функційна мова програмування, для декларативного створення браузерних застосунків з графічним інтерфейсом користувача. Elm використовує стиль функційного реактивного програмування та чисто функційне програмування для того щоб сконструювати інтерфейс користувача унеможливлюючи деструктивні зміни.

Згідно до ідеї творців і не дивлячись на те, що Facebook предоставив реалізацію Flux як доповнення до React, Flux не є ще одним веб-фреймворком, а надає архітектурне рішення.

Redux пропонує думати про за стосунок, яяк про початковий стан, який можна модифікувати послідовністю дій, що являється справді правильним підходом для складних веб-застосунків та відкриває багато можливостей.

Головною відмінністю Redux від інших підходів є той факт, що всі стани зберігаються в одному місці і застосунок будується за однонапрямленим потоком даних.

Однонапрямдений потік даних кардинально відрізняється від MVC, де C—V і C—M зазвичай двонапрямлені. Однонапрямлений потік спрощує розуміння застосунку. Місце, в якому зберігаються дані, називається «стор».

Другий принцип — стани за стосунку не можна змінювати напряму. Тобто ось так store.number += 1 робити не можна. Щоб змінити стор, необхідно згенерувати подію з описом зміни. Це можна зробити за допомогою store.dispatch(). Об’єкт, який передається в dispatch(), називається «екшн» (action). Це звичайний JavaScript-об’єкт. Він може включати в себе будь-які

поля, які будуть описувати зміну, єдине обов'язкове поле — type. Redux забороняє напряму змінювати стан тому, що необхідно слідкувати за змінами щоб при змінах стора оновлювати вигляд.

Третій принцип — вся зміни оброблюються функцією, яка називається «редьюсер» і яка повинна бути чистою.

Чиста функція — це функція, яка завжди для одного і того ж вхідного значення повертає однаковий результат і не змінює нічого ззовні себе. Синус — чиста функція тому, що для одного й того ж кута вона завжди повертає одне й те ж значення.

parseInt() — чиста.

$.ajax() — не чиста тому, що в різні моменти часу вона може повернути різні значення.

```
function reducer(state, action) {
    switch(action.type) {
        case 'INCREMENT' : return {number: state.number + action.by}
        default:  return state
    }
}
```

Ця функція, яка обробляє екшни, називається редьюсером. Вона обробляє кожний екшн, який відбувається у застосунку. Функція приймає на вхід теперішній стан і екшн, який щойно відбувся і для кожного типу екшена змінює стан таким чином, яким вважає за потрібне. Варто звернути увагу на те, що стан тут не змінюється напряму. Замість цього повертається новий стан, яким Redux вже сам замінює старий.

Відповідно до цих принципів, Redux – застосунок будується за схемою, вказаною на рисунку 2.2:

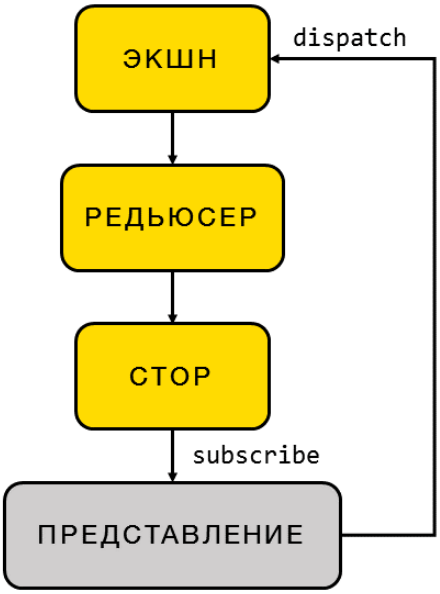

Рисунок 2.2 – Схема потоків даних в Redux-архітектурі

Але у такого підходу є і недоліки, як, наприклад, робота із сайд-ефектами.

Сайд-ефекти — це коли ми змінюємо глобальну змінну, робимо запит на сервер або ж пишемо щось в лог, тобто робимо якісь дії, що не стосуються роботи функції.

Проблема в тому, що в цій схемі немає місця сайд-ефектам. Єдине місце, де відбуваються події – це редьюсер, а він має бути чистою функцією. Тут на допомогу приходить middleware. Мідлвейр знаходиться між кодом, який діспатчить екшени (тобто генерує події), і редьюсером (тобто функцією, яка перетворює в новий стан), пропускає через себе всі виникаючі екшени, змінює їх (якщо вважає за потрібне) і відправляє далі – або не відправляє. Особливість мідлвейра в тому, що він бути чистою функцією не має бути і тому він може робити сайд-ефекти типу запитів на сервер або чого завгодно.

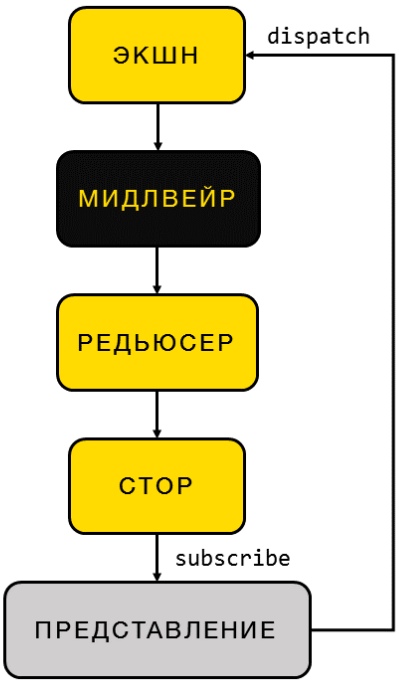

Рисунок 2.3 – Розміщення мідлвейру в Redux-архітектурі

Насправді, мідлвейрів може бути декілька і тоді вони організовуються ланцюжком.

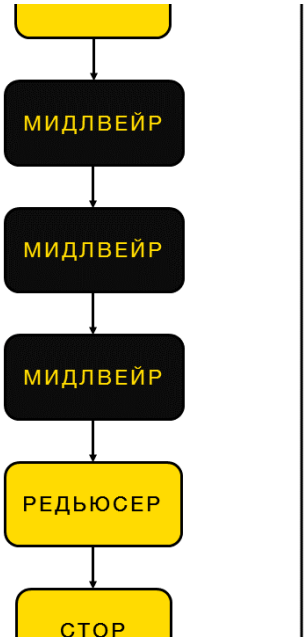

Рисунок 2.4 – Ланцюжок мідлвейрів у Redux-архітектурі

Фактично, мідлвейр – це просто функція, яка приймає поточний стор, який йде наступним після мідлвейру в ланцюжку, і поточний екшн і щось робить з ними.

Const logMiddleware =

```
Store => next => action => {
        Console.log(action);
        Next(action);
}
Const store = createStore(reducer,
applyMiddleware(logMiddleware))
```

Мідлвейри підключаються в момент створення стора. createStore і applyMiddleware — це методи Redux. Пропоную розглянути існуючі рішення використання мідлвейру для роботи із сайд-ефектами. Варто зазначити, що мідлвейри створюються суспільством і не входять в поставку Redux.

Є багато методів організацій сайд-ефектів. Розглянемо два найпопулярніших. Перший – Redux Thunk, який було використано в даній роботі. Він потрібен тому, що класичний Redux вміє працювати тільки з екшенами, які являються простими об’єктами. Застосовуючи мідлвейр Redux Thunk таким чином ми отримуємо можливість замість простих об’єктів диспатчити функції. Коли ми диспатчимо функцію, Redux Thunk викликає її та передає в неї метод dispatch стора, який потім можна буде викликати тоді, коли необхідно. Такі функції називаються Thunk-ами.

Ще за допомогою Redux Thunk можна диспатчити декілька екшенів поспіль, щоб виконати логаут і потім очистити стор. Можна диспатчити екшени за таймером, щоб відображати користувачу модальні вікна або ж закінчувати певні дії по закінченню часу. Redux Thunk зручний для простих сайд-ефектів. Коли необхідно організувати складний потік сайд-ефектів, він стає незручним. В цьому випадку на допомогу приходить Redux Saga.

Після розробки архітектури програмного забезпечення було створено структурну схему класів програмного забезпечення, яку наведено на КПІ.ІП-4410.045490–07–99.СС.

Для розробки програмного інтерфейсу системи для оцінки ризиків було прийнято використовувати інструменти для конфігурації, уніфікації та стандартизації коду.

JSlint та TsLint [7]

JSlint - це статичний аналізатор коду з веб-інтерфейсом для програм мовою JavaScript, перевіряє їх відповідність стандартам оформлення коду, розроблений Дагласом Крокфордом. JSLint бере вихідний код JavaScript і сканує його. Якщо він виявить проблему, він повертає повідомлення, що описує проблему, та приблизне місцезнаходження в межах джерела. Проблема не обов'язково є синтаксичною помилкою, хоча зазвичай це трапляється саме так. JSLint розглядає деякі стильові конвенції, а також структурні проблеми.

JSLint визначає професійну підмножину JavaScript, більш сувору мову, ніж та, що визначається Стандартом мов програмування є ECMAScript.

JavaScript є недбалою мовою, але всередині прихована глибока елегантна, краща мова. JSLint допомагає вам програмувати на цій кращій мові та уникнути більшості схилів. JSLint відкидає програми, які браузери приймуть, тому що JSLint стурбована якістю вашого коду.

Даний статичний аналізатор використовується на стороні серверу де для *.js файлів підчас розобки та перевірки при зборці.

TSLint - це розширюваний інструмент статичного аналізу, який перевіряє TypeScript-код для помилок читання, ремонтопридатності та функціональності. Він широко підтримується сучасними редакторами та системами побудови, і його можна налаштувати за допомогою власних правил, конфігурацій та формати.

Збірка системи

Для збірки системи було використано такі технології:

1) Docker [8]

Docker було обрано за для того, щоб система збирала серверну частину та клієнтську частину завжди однаково, не зважаючи на якій машині була виконана збірка.

Docker складається з двох процесів:

– демона Docker, який запускається на гостьовій машині (якщо це Лінукс), або всередині VirtualBox середовища boot2docker (якщо це Windows або OS X);

– клієнта, через який можна взаємодіяти з демоном;

– образ Docker (англ. Docker image) - містить операційну систему, застосунок і всі його залежності. Образи в Docker складаються з шарів. Якщо нам треба образ з веб-сервером, то ми беремо за основу образ з дистрибутивом операційної системи, додаємо залежність - веб-сервер, і записуємо це як новий образ, який матиме два шари - один з ОС, наступний з веб-сервером;

– контейнер Docker - це запущений образ. Контейнери Docker можна запускати, спиняти, переміщувати і видаляти. Також можна зробити docker commit контейнера, що створить образ з поточного стану контейнера.

Основні можливості Docker:

– можливість розміщення в ізольованому оточенні різнорідної начинки, що включає різні комбінації виконуваних файлів, бібліотек, файлів конфігурації, скриптів, файлів jar, gem, tar тощо;

– підтримка роботи на будь-якому комп’ютері на базі архітектури x86_64 з системою на базі ядра Linux, починаючи від ноутбуків, закінчуючи серверами та віртуальними машинами. Можливість роботи поверх немодифікованих сучасних ядер Linux (без накладення патчів) і в штатних оточеннях всіх великих дистрибутивів Linux, включаючи Fedora, RHEL, Ubuntu, Debian, SUSE, Gentoo і Arch;

– використання легковагих контейнерів для ізоляції процесів від інших процесів і основної системи;

– оскільки контейнери використовують свою власну самодостатню файлову систему, не важливо де, коли і в якому оточенні вони запускаються;

– ізоляція на рівні файлової системи: кожен процес виконується у повністю окремій кореневій ФС;

– ізоляція ресурсів: споживання системних ресурсів, таких як витрата пам'яті і навантаження на CPU, можуть обмежуватися окремо для кожного контейнера з використанням cgroups;

– ізоляція на рівні мережі: кожен ізольований процес має доступ тільки до пов'язаного з контейнером мережевого простору імен, включаючи віртуальний мережевий інтерфейс і прив'язані до нього IP-адреси;

– коренева файлова система для контейнерів створюється з використанням механізму copy-on-write[en] (окремо зберігаються тільки змінені і нові дані), що дозволяє прискорити розгортання, знижує витрату пам'яті і економить дисковий простір;

– всі стандартні потоки (stdout/stderr) кожного виконуваного в контейнері процесу накопичуються і зберігаються у вигляді логу.

2) Webpack[9]

Для збірки клієнтської частини було використано WebPack.

Webpack є модулем з відкритим вихідним кодом JavaScript. Webpack є модульним пакетом, в першу чергу, для JavaScript, але може трансформувати інтерфейсні ресурси, такі як HTML, CSS і зображення, якщо відповідні плагіни включені. Webpack приймає модулі з залежностями і генерує статичні засоби, що представляють ці модулі.

Webpack приймає залежності і генерує графік залежностей, що дозволяє використовувати модульний підхід для розробки веб-додатків. Він може бути використаний з командного рядка або може бути налаштований за допомогою конфігураційного файлу, який називається webpack.config.js. Цей файл використовується для визначення навантажувачів, плагінів тощо для проекту.

(Webpack є дуже розширюваним через навантажувачі, які дозволяють розробникам писати користувальницькі завдання, які вони хочуть виконувати при об'єднанні файлів разом.) Інструмент з назвою createapp.dev спрощує процес створення цього конфігураційного файлу. На рисунку 2.5 наглядно зображено роботу Webpack, при збірці файлів з багатьма залежностями та розширеннями файлів.

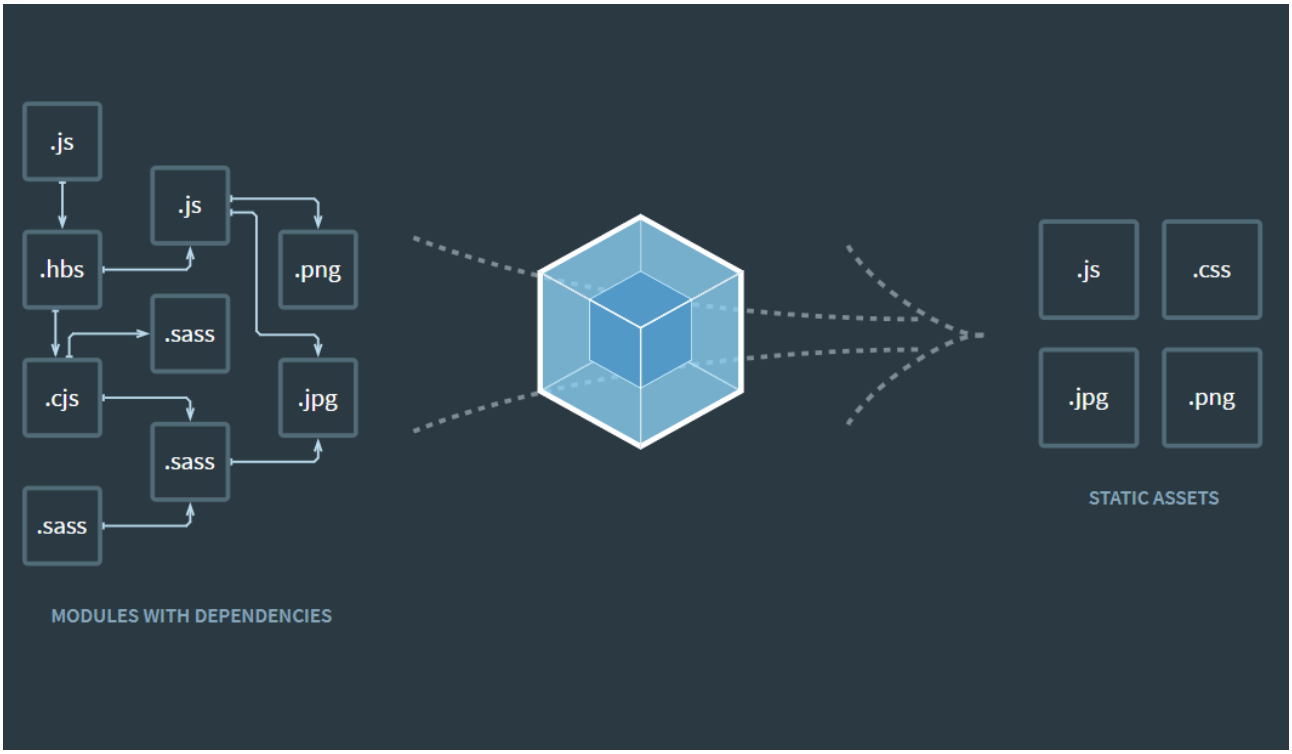

Рисунок 2.5 – Наглядний приклад роботи WebPack

Специфікація API (Swagger)[10]

Користувальницький інтерфейс Swagger дозволяє візуалізувати та взаємодіяти з ресурсами API. Він автоматично генерує OpenAPI Специфікацію, а візуальна документація полегшує розуміння користувача.

Специфікація OpenAPI, раніше відома як Специфікація Swagger, є світовим стандартом для визначення RESTful інтерфейсів. OAS дозволяє розробникам розробляти технологично-агностичний API інтерфейс, що є основою для розробки та використання їх API.

VCS & Gitlab & GIT[11]

Система керування версіями (англ. source code management, SCM) — програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D-моделі, текстового документу тощо. Це інструмент, який дозволяє одночасно, не заважаючи один одному, проводити роботу над груповими проектами.

Системи керування версіями зазвичай використовуються при розробці програмного забезпечення для відстеження, документування та контролю над поступовими змінами в електронних документах: у сирцевому коді застосунків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють декілька людей.

Кожна версія позначається унікальною цифрою чи літерою, зміни документу занотовуються. Зазвичай також зберігаються дані про автора зробленої зміни та її час.

Інструменти для контролю версій входять до складу багатьох інтегрованих середовищ розробки.

Існують два основні типи систем керування версіями: з централізованим сховищем та розподіленим.

Система збереження історії редагувань статей, що застосовується у Вікіпедії є прикладом системи керування версіями. Git - це безкоштовна та відкрита система розподіленої версії версій, призначена для обробки всього від малих до дуже великих проектів із швидкістю та ефективністю.

Система контролю дозволяє зберігати попередні версії файлів та завантажувати їх за потребою. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки. Місце зберігання даних файлів називають репозиторієм. В середині кожного з репозиторіїв можуть бути створені паралельні лінії розробки — гілки.

Гілки зазвичай використовують для зберігання експериментальних, незавершених(alpha, beta) та повністю робочих версій проекту(final).

Більшість систем контролю версії дозволяють кожному з об'єктів присвоювати теги, за допомогою яких можна формувати нові гілки та репозиторії.

Використання системи контролю версії є необхідним для роботи над великими проектами, над якими одночасно працює велика кількість розробників. Системи контролю версії надають ряд додаткових можливостей:

– можливість створення різних варіантів одного документу;

– документування всіх змін (коли ким було змінено/додано, хто який рядок змінив);

– реалізує функцію контролю доступу користувачів до файлів. Є можливість його обмеження;

– дозволяє створювати документацію проекту з поетапним записом змін в залежності від версії;

– дозволяє давати пояснення до змін та документувати їх.

Git дозволяє вам мати кілька місцевих відділень (branch), які можуть бути цілком незалежними один від одного. Основою розробки є створення, об'єднання та видалення цих ліній розвитку. Такт управління розробкою, перевіркою коду.

GitLab - це програма, використовується для роботи на всіх етапах життєвого циклу DevOps для продуктів, розробників, QA, безпеки та операційних груп, щоб одночасно працювати на одному проекті. GitLab дозволяє командам співпрацювати та працювати з однієї бесіди, замість того, щоб керувати кількома потоками за різними інструментами. GitLab надає командам єдиний магазин даних, один користувальницький інтерфейс і одна модель дозволу в життєвому циклі DevOps, що дозволяє командам співпрацювати, значно скорочуючи час циклу та зосереджуючись виключно на створенні програмного продукту.

Поточний проект зіставляєтеся з 2 проектів це сервер і мобільній додаток які роздільно підтримується GIT системою управління версіями.

Вони в свою чергу розміщені на сервері та для зручності розробки, огляд коду та управління процесом розробки, використовують програмне забезпечення GitLab.

Unit & e2e Testing

Одиничне тестування, або модульне тестування (англ. unit testing) - процес в програмуванні, що дозволяє перевірити на коректність одиниці вихідного коду, набори з одного або більше програмних модулів разом з відповідними керуючими даними, процедурами використання і обробки.

Ідея полягає в тому, щоб писати тести для кожної нетривіальною функції або методу. Це дозволяє досить швидко перевірити, чи не призвело чергову зміну коду до регресії, тобто до появи помилок в уже тестувати місцях програми, а також полегшує виявлення і усунення таких помилок.

Для реалізації unit testing використовуються інструменти enzyme.

Enzyme - це утиліта тестування JavaScript для React, яка полегшує тестування виводу компонентів React. Можна також керувати, проходити, і в деяких ситуаціях імітувати виконання коду з виведенням результату. [4]

Jenkins

Jenkins - проект для безперервної інтеграції з відкритим вихідним кодом, написаний на Java. Був ответвлён від проекту Hudson, що належить компанії Oracle. Поширюється під ліцензією MIT.

Автоматизувати частину процесу розробки програмного забезпечення, в якому не обов'язково участь людини, забезпечуючи функції безперервної інтеграції. Працює всередині в сервлет-контейнер. Підтримує інструменти системи управління версіями, включаючи AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase і RTC. Збірка може бути запущена різними способами, наприклад, за подією фіксації змін в системі управління версіями, за розкладом, по запиту на певний URL, після завершення другого збірки в черзі.

Можливості Jenkins можна розширювати за допомогою плагінів.

В поточному проекті використовується програмне забезпечення Jenkins для автоматичної збірки проекту на сервері та використовуються наступні кроки:

– перевірка Tslint, jslint;
– перевірка Unit tests за допомогою вбудованої команди ng test;
– E2E tests;
– збірка проекту;
– автоматична доставка та оновлення серверу.

Збірка відбувається за користувацьким ініціюванням або через ініціюванням програмою GitLab по пуш на гілку мастер через (webhook).

Code review (Sonarqube) [12]

Перегляд коду (англ. Code review) або інспекція коду (англ. Code inspection) - систематична перевірка вихідного коду програми з метою виявлення та виправлення помилок, які залишилися непоміченими в початковій фазі розробки. Метою перегляду є поліпшення якості програмного продукту і вдосконалення навичок розробника.

В процесі інспекції коду можуть бути знайдені і усунені такі проблеми, як помилки у форматуванні рядків, стан гонки, витік пам'яті і переповнення буфера, що покращує безпеку програмного продукту. Системи контролю версій дають можливість проведення спільної інспекції коду. Крім того, існують спеціальні інструментальні засоби для спільної інспекції коду.

Для організації роботи та перевірки коду було залучено програмний продукт SonarQube.

SonarQube – це платформа для перевірки коду на якість за правилами, заснованим на угодах і стандартах. Підтримує більше 20 різних мов програмування.

Програмне забезпечення для автоматизованої інспекції коду спрощує завдання перегляду великих шматків коду, систематично скануючи його на предмет виявлення найбільш відомих вразливостей.

SonarQube вимірює якість програмного коду відповідно до семи показниками (і відповідними метриками) якості програмного забезпечення, які розробники називають англ. Seven Axes of Quality:

– потенційні помилки;
– стиль програмування;
– тести;
– повторення ділянок коду;
– коментарі;
– архітектура та проектування;
– складність.

Так, тести оцінюються не тільки з точки зору успішності виконання, але і по тестовому покриттю вихідного коду.

2.3 Конструювання програмного забезпечення

Опис класів + опис вхідних/вихідних даних

2.4 Аналіз безпеки даних

Для захисту даних, було використано кращі практики написання програмного коду та рекомендації від розробників expressjs[8] та angular[9].

Застосунок було розроблено за допомогою фреймворку Angular, а в ньому закладені певні захисні системи. Найважливіша з них – захист від атак типу XSS.

XSS (англ. Cross Site Scripting — «міжсайтовий скриптинг») — тип вразливості інтерактивних інформаційних систем у вебі. XSS виникає, коли на сторінки, які були згенеровані сервером, з якоїсь причини потрапляють користувацькі скрипти. Специфіка подібних атак полягає в тому, що замість безпосередньої атаки сервера зловмисники використовують вразливий сервер для атаки на користувача.

Для терміну використовують скорочення «XSS», щоб не було плутанини з каскадними таблицями стилів (абревіатура «CSS»).

Довгий час програмісти не приділяли їм належної уваги, вважаючи їх безпечними. Однак ця думка помилкова: на сторінці або в HTTP-Cookie можуть бути досить вразливі дані (наприклад, ідентифікатор сесії адміністратора). На популярному сайті скрипт може влаштувати DoS-атаку.[7]

Використовуючи Angular, ми захищені від даного типу атак за замовчуванням. Angular систематично блокує будь-які XSS помилки, він приймає всі значення, як підозрілі за замовчуванням. Коли значення додається в DOM, через шаблон, властивість, атрибут, стиль або інтерполяцію. Він видаляє та уникає всі підозрілі значення.

2.5 Висновки по розділу

В даному розділі були описані всі технічні і програмні засоби, що використовувалися для створення системи. Описані технології, завдяки яким була розроблена система. Для реалізації даної системи було використано шаблон MVC.

Для розробки програмного продукту - системи вивчення абетки для дітей дошкільного віку із застосуванням ігрового підходу абетки на ionic та node.js було прийнято рішення використовувати інструменти для конфігурації, уніфікації та стандартизації коду. По-перше, було застосовано статичний аналізатор, який використовується на стороні серверу де для *.js файлів підчас розробки та перевірки при зборці – JSLint. Також необхідно зазначити, що для мобільного продукту було погоджено використовувати строгу типізацію та для усіх *.ts файлів перевірку коду через TSLint аналізатора.

Також було використано відому систему контролю версій Git. Git - це безкоштовна та відкрита система розподіленої версії версій, призначена для обробки всього від малих до дуже великих проектів із швидкістю та ефективністю. Для організації роботи та перевірки коду було залучено програмний продукт SonarQube.

Для захисту даних, було використано кращі практики написання програмного коду та рекомендації від розробників expressjs та angular.

Було описано функції, таблиці бази даних, масиви інформації, наведено їх детальний розбір.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Співвідношення під характеристик і атрибутів (метрик):

– **функціональна придатність** – функціональна адекватність:

X = 1 – A/B,

де A – к-сть функцій, в яких наявні функціональні проблеми,

B – загальна кількість функцій;

– **захищеність** – контроль доступу та змін:

X = A/B,

де A – кількість обізнаних типів доступу до системи,

B – загальна кількість нелегальних типів доступу до системи;

– **завершеність** – кількість помилок відносно кількості тестів:

X = A1/A2,

де A1 – кількість помічених помилок,

A2 – загальна кількість тестів;

– **відновлюваність** – середній час недоступності:

де X = T/N,

T – загальна кількість часу, поки система не доступна,

N – кількість помічених вимкнень системи;

– **зрозумілість** – завершеність опису:

де X = A/B,

A – кількість зрозумілих після читання описів функцій,

B – загальна кількість функцій ;

– **стабільність** – коефіцієнт успішності зміни:

X = N/T,

де N – кількість помилок , які отримує користувач після зміни ПЗ,

T – час, який пройшов з моменту зміни ПЗ;

Таблиця 3.1 – Функціональні вимоги і їх відповідність під характеристикам

Функціональна вимога	**Під характеристики**
Адаптивність інтерфейсу системи на всіх платформах	Функціональна придатність, зрозумілість, завершеність, стабільність.
Авторизація	Захищеність, завершеність, зрозумілість, стабільність.
Коректне відображення вікон системи	Функціональна придатність, зрозумілість, завершеність, стабільність.
Створення користувачів (власників, підписників)	Функціональна придатність, зрозумілість, завершеність, керованість, стабільність.
Робота з типами оцінок	Функціональна придатність, завершеність, зрозумілість, стабільність.
Генерація звіту	Функціональна придатність, відновлюваність, стабільність, захищеність, завершеність.

Таблиця 3.2 – Нефункціональні вимоги і їх відповідність під характеристикам

Нефункціональні вимоги	**Під характеристики**
Застосування	Стабільність, завершеність.
Продуктивність	Стабільність, завершеність.
Зручність використання	Стабільність, завершеність, зрозумілість.
Надійність виконання	Захищеність, відновлюваність.

Експлуатаційні якості:

– **результативність** – завершеність задач:

X = A/B,

де A – кількість виконаних задач,

B – кількість задач, яких була спроба виконати;

– **продуктивність** – витрата ресурсів(часу) для досягнення необхідної мети:

X = Ta/Tb,

де Ta – час ефективної роботи,

Tb – загальний час роботи;

– **безпека** – рівень нанесення збитків ПЗ:

X = 1 – A/B,

де A – кількість помічених збитків ПЗ,

B – загальна кількість використання;

– **задоволеність** – рівень задоволення використання ПЗ в заданому контексті:

X = A/B,

де A – кількість використання деякого методу чи функції,

B – кількість задуманого використання цього методу чи функції.

Таблиця 3.3 – Відповідність експлуатаційних характеристик та внутрішнім характеристикам

Експлуатаційні характеристики	Внутрішні
Продуктивність	Стабільність
Результативність	Завершеність, функціональна придатність
Безпека	Захищеність, відновлюваність
Задоволеність	Зрозумілість

Дані для тестування:

а) 180 – кількість функцій;

б) 20 – кількість функцій з проблемами;

в) 2 – кількість обізнаних типів доступу до системи;

г) 1 – кількість нелегальних типів доступу;

д) 15 – кількість помічених помилок;

е) 25 – загальна кількість тестів;

ж) 10с – час поки система недоступна;

з) 2 – помічених вимкнень системи;

и) 120 – зрозумілих описів функцій;

к) 58 – кількість виконаних задач;

л) 60 - кількість задач, яких була спроба виконати;

м) 60 – час ефективної роботи;

н) 70 – загальний час роботи;

о) 0 - кількість помічених збитків ПЗ;

п) 1 - загальна кількість використання;

р) 27 - кількість використання деякого методу чи функції;

с) 27 - кількість задуманого використання цього методу чи функції.

Раніше були визначені наступні характеристики якості:

а) функціональність:

1) функціональна придатність;

б) надійність:

1) захищеність;

2) відновлюваність;

в) зручність застосування:

1) зрозумілість;

г) супроводжуваність:

1) стабільність;

А також для експлуатаційної якості:

– результативність;

– продуктивність;

– безпечність;

– задоволеність.

Розрахуємо метрики Холстеда для підхарактеристик зручності застосування.

– **зрозумілість** :

X = A/B,

A= 180;

B = 120;

Результат = 1,5;

– **функціональна придатність**:

X = 1 – A/B,

A = 20;

B = 180;

Результат = 0,88;

– **захищеність**:

X = A/B,

A = 2;

B = 1;

Результат = 2;

– **завершеність** – кількість помилок відносно кількості тестів:

X = A1/A2,

A1 = 15;

A2 = 25;

Результат = 0,6;

– **відновлюваність** – середній час недоступності:

де X = T/N,

T = 10,

N = 2;

Результат = 5.

Підрахуємо результати експлуатаційних якостей:

– **результативність** :

X = A/B,

A = 58;

B = 60;

Результат = 0,97.

– **продуктивність**:

X = Ta/Tb,

Ta = 60;

Tb = 70;

Результат = 0,86.

– **безпека**:

X = 1 – A/B,

A = 0;

B = 1;

Результат = 1.

– **задоволеність**:

X = A/B,

A = 27;

B = 27;

Результат = 1 .

3.2 Опис процесів тестування

Процес тестування програмного забезпечення, являється важливим етапом життєвого циклу розробки. Цей процес дозволяє не лише перевірити якість готового продукту, а й перевірити виконання, поставлених вимог та функцій, які мають бути реалізовані. З допомогою тестування , можна охопити

різні елементи застосунку, опираючись на наявні часові та людські ресурси. Та виявити критичні проблеми, до здійснення впровадження.

Для тестування створеної системи було обрано метод функціонального тестування, за допомогою якого можна перевірити правильність виконання поставлених задач.

Було вирішено перевірити наступні основні функції системи:

– авторизація у системі (Таблиця 3.4);
– створення фактора (Таблиця 3.5);
– створення загрози (Таблиця 3.6);
– створення типу оцінки (Таблиця 3.7);
– конфігурування методології (Таблиця 3.8);
– створення підписника (Таблиця 3.9);
– редагування підписника зі статусом «Очікує підтвердження» (Таблиця 3.10);
– редагування підписника зі статусом «Активний» (Таблиця 3.11).

Тестування буде мати задовільний результат при:

– правильності збережені даних в базу даних;
– наданні повідомлення помилки при введені некоректних даних;
– коректними вихідними даними функції, що тестується;
– коректному відображені інтерфейсу системи.

3.3 Опис контрольного прикладу

В процесі тестування була перевірена вся функціональність системи. Послідовно були перевірені всі варіанти зазначені вище функції системи, результати представлені у відповідних таблицях:

Таблиця 3.4 – Перевірка можливості авторизації у системі

Мета тесту	Перевірка можливості авторизації у системі.
Початковий стан	Відкрито авторизаційну форму системи.
Вхідні дані	Ел. адреса та пароль користувача

Схема проведення тесту	Відкрити веб-сайт; на відображеній сторінці заповнити форму авторизації (ввести ел. адресу та правильний пароль); натиснути «Увійти» або кнопку «Enter».
Очікуваний результат	Система успішно авторизує користувача, відображує головну сторінку з опціями відповідних варіантів доступу.
Стан програмного продукту після проведення випробувань	Відкрита головна сторінка системи, користувач авторизований.

Таблиця 3.5. – Перевірка можливості створення фактора

Мета тесту	Перевірка можливості створення фактора
Початковий стан	Відкрита головна сторінка.
Вхідні дані	Відсутні.
Схема проведення тесту	Відкрити список факторів; натиснути кнопку «Додати фактор»; ввести дані; натиснути кнопку «Зберегти».
Очікуваний результат	Система успішно зберігає створений фактор.
Стан програмного продукту після проведення випробувань	Створений фактор відображається у таблиці.

Таблиця 3.6. – Перевірка можливості створення загрози

Мета тесту	Перевірка можливості створення загрози
Початковий стан	Відкрита головна сторінка.
Вхідні дані	Відсутні.
Схема проведення тесту	Відкрити список загроз; натиснути кнопку

	«Додати загрозу»; ввести дані; натиснути кнопку «Зберегти».
Очікуваний результат	Система успішно зберігає створену загрозу.
Стан програмного продукту після проведення випробувань	Створена загроза відображається у таблиці.

Таблиця 3.7 – Перевірка можливості створення типу оцінки

Мета тесту	Перевірка можливості створення типу оцінки
Початковий стан	Відкрита головна сторінка.
Вхідні дані	Відсутні.
Схема проведення тесту	Відкрити список типів оцінок; натиснути кнопку «Додати тип оцінки»; ввести дані; натиснути кнопку «Зберегти».
Очікуваний результат	Система успішно зберігає створений тип оцінки.
Стан програмного продукту після проведення випробувань	Створений тип оцінки відображається у таблиці.

Таблиця 3.8 – Перевірка можливості конфігурації методології

Мета тесту	Перевірка можливості конфігурування методології.
Початковий стан	Відкрита сторінка методологій.
Вхідні дані	Наявна хоча б одна існуюча методологія.
Схема проведення тесту	Обрати методологію; відкрити будь-яку вкладку; натиснути кнопку «Редагувати»; змінити будь-які дані; натиснути кнопку «Зберегти».
Очікуваний результат	Зміни в даних таблиці було збережено.
Стан програмного	Зміни відображаються в інформації про

продукту після проведення випробувань	методологію.

Таблиця 3.9 – Перевірка можливості створення підписника

Мета тесту	Перевірка можливості створення підписника
Початковий стан	Відкрита головна сторінка.
Вхідні дані	Відсутні.
Схема проведення тесту	Відкрити список підписників; натиснути кнопку «Додати підписника»; ввести дані нового підписника; натиснути кнопку «Запросити».
Очікуваний результат	Система надсилає запрошення новому підписнику та зберігає його дані.
Стан програмного продукту після проведення випробувань	Створений підписник відображається у списку.

Таблиця 3.10 – Перевірка редагування підписника зі статусом «Очікує підтвердження»

Мета тесту	Перевірка редагування підписника зі статусом «Очікує підтвердження»
Початковий стан	Відкрита сторінка підписників.
Вхідні дані	В списку присутній хоча б один підписник зі статусом «Очікує підтвердження».
Схема проведення тесту	Обрати підписника; натиснути кнопку «Редагувати»;змінити будь-які дані; натиснути кнопку «Зберегти».
Очікуваний результат	Система успішно зберігає зміни в інформації про підписника. Якщо було змінено ел. адресу, нове запрошення відправлене на нову адресу.

Стан програмного продукту після проведення випробувань	Змінений підписник відображається з новими даними.

Таблиця 3.11 – Перевірка редагування підписника зі статусом «Активний»

Мета тесту	Перевірка редагування підписника зі статусом «Активний»
Початковий стан	Відкрита сторінка підписників.
Вхідні дані	В списку присутній хоча б один підписник зі статусом «Активний».
Схема проведення тесту	Обрати підписника; натиснути кнопку «Редагувати»;змінити доступні до зміни; натиснути кнопку «Зберегти».
Очікуваний результат	Система успішно зберігає зміни в інформації про підписника.
Стан програмного продукту після проведення випробувань	Змінений підписник відображається з новими даними.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Описана в дипломній роботі система працює на більшості популярних пристроїв: мобільних девайсах, планшетах та браузерах (Google Chrome, Opera,Mozilla Firefox, Safari).

Система буде розгорнута на веб-хості, за посиланням https://risk-assessment.com/

Системний адміністратор має доступ до системи одразу. Для входу в систему йому необхідно ввести дані, які спочатку було додано до бази даних системи.

Власники отримують доступ до системи після того, як системний адміністратор створює профіль кожного власника та відправляє запрошення з інтерфейсу вкладки «Додати власника». Власнику необхідно отримати, прийняти запрошення та перейти за посиланням, після чого використати дані для входу (ел. адресу та пароль).

Підписники можуть таким же чином отримувати доступ до системи. Профіль підписника може бути створений системним адміністратором або ж власником. Підписнику також відправляється запрошення до системи за вказаною електронною адресою, після чого він приймає запрошення та використовує ел. адресу та пароль в подальшому для входу в систему.

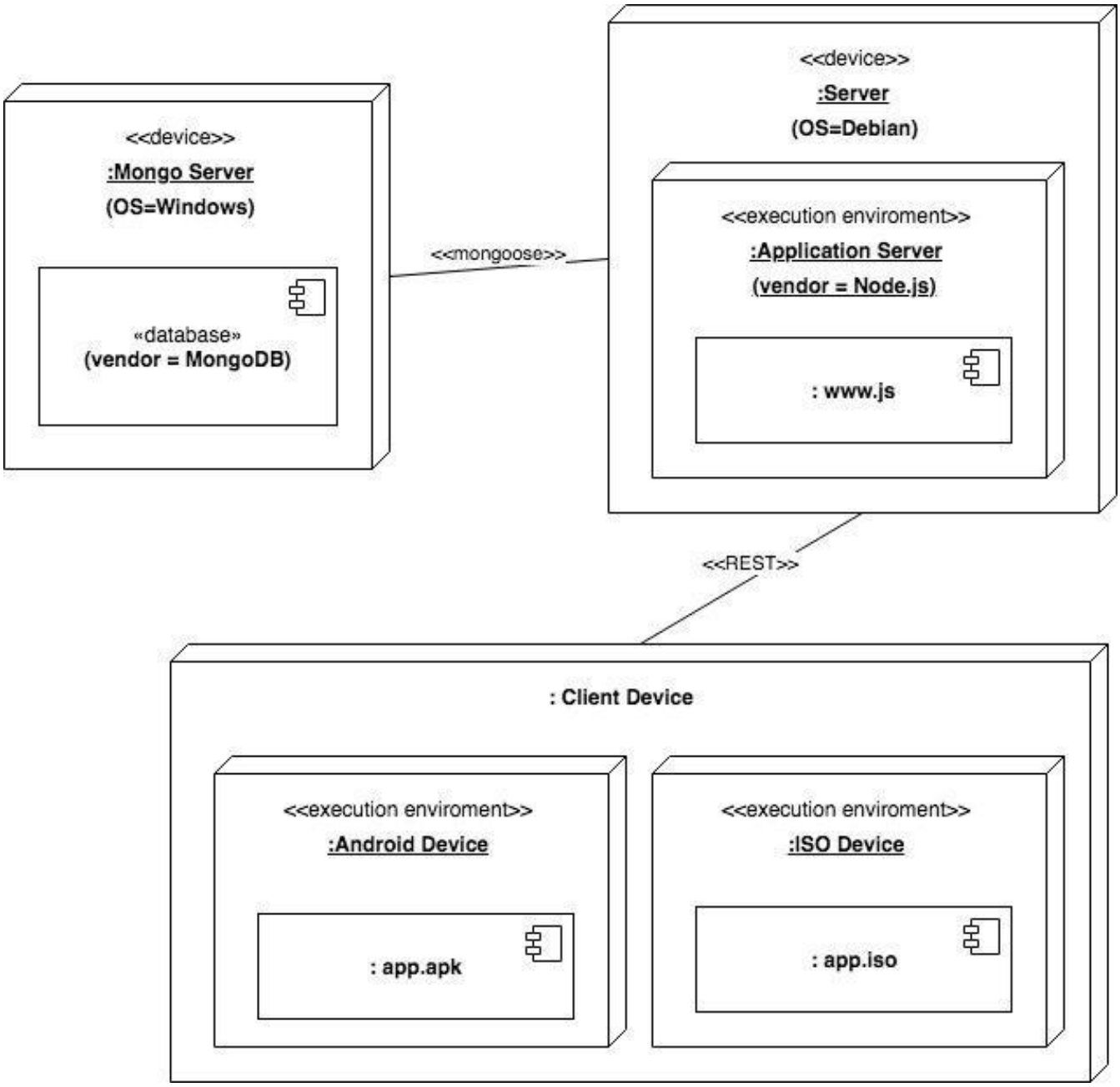

Рисунок 4.1 – Структурна схема розгортання програмного комплексу

4.2 Робота з програмним забезпеченням

Супровід програмного забезпечення описано в документі КПІ.ІП-4410.045440-06-99.

ВИСНОВКИ

Метою даної роботи була розробка програмного інтерфейсу для системи оцінки ризиків. Головною метою даної системи є полегшити та зробити більш доступним використання методологій та видів оцінки ризиків власникам та клієнтам аудиторських компаній.

У першому розділі було наведено приклади готових існуючих рішень, описано їх головні переваги та недоліки, визначено основні функціональні вимоги, визначено основні нефункціональні вимоги, визначено основні функції модуля, створену діаграму вимог та побудовано таблицю варіантів використання даного додатку.

В другому розділі описано архітектуру програмного продукту, основні принципи, які було використано при розробці, технології, за допомогою яких розробили продукт, побудовано діаграму класів та описано всі класи даної діаграми, які відповідають завдання модуля, побудовано систему зберігання даних користувача та деяких системних даних, описано систему захисту даних, яка використовується в системі.

У третьому розділі було проведено тестування етапів застосунку, з детальним описом кожного тесту та результатів його виконання. За результатами тестування, застосунок виконує найважливіші вимоги, але деякі функції мають бути пропрацьовано краще. Було проведено ретельний аналіз якості програмного застосунку, та виявлено деякі помилки, на які ми звернули увагу, для подальшого підвищення якості.

В четвертому розділі наведено інструкцію користувача по використанню системи. Також було детально описано весь функціонал доступний для використання та показано приклади його застосування з детальними поясненнями. Було описано розгортання програмного продукту, що треба для роботи з ним та було побудовано структурну схему розгортання програмного продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1) MongoDB - Режим доступа: https://mongodb.org/
2) User Experience - Режим доступа: https://en.wikipedia.org/wiki/User_experience
3) NoSQL - Режим доступа: https://uk.wikipedia.org/wiki/NoSQL
4) AngularJS - Режим доступа: https://uk.wikipedia.org/wiki/AngularJS
5) XSS - Режим доступа: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
6) Enzyme - https://airbnb.io/enzyme/
7) Elm - https://ua.wikipedia.org/wiki/Elm_(мова_програмування)
8) JSLint - Режим доступа: https://uk.wikipedia.org/wiki/JSLint
9) Docker - https://uk.wikipedia.org/wiki/Docker
10) Webpack - https://en.wikipedia.org/wiki/Webpack
11) API Swagger – Режим доступа: https://uk.wikipedia.org/wiki/OpenAPI
12) VCS - https://uk.wikipedia.org/wiki/Система_керування_версіями
13) SonarQube - Режим доступа: https://dou.ua/lenta/articles/coverage-report/